University of Cape Town Capstone Project:

LEAP - Let's go phishing!

Computer Science Department
Science Faculty, UCT

Authors

Mxolisi Vilakazi

Rikard Eide

Phumlile Sopela

22 September 2014

Abstract

This report explains in detail how a simple utility Android application - a flashlight application, can exploit a user's privacy on their mobile device by scanning through a user's messaging application and extract banking SMS messages and send them out to a malicious party. This utility application commits the act of phishing. This report also explains how another Android application can be used to raise awareness of applications installed on the user's mobile device that send out data. Both of these applications were designed and developed for the Computer Science department of UCT.

Table of contents

1. Introduction
2. Pre-study and research
2.1 Phishing
2.2 Method
2.2 Existing solutions
2.3 Development tools
3. Requirements captured
3.1 Functional requirements
3.1.1 Phishlight requirements
3.1.2 Phishlight Use Cases
3.1.3 Phishlight server requirements
3.1.4 Phishlight Server Use Cases
3.1.5 Phishguard requirements
3.1.6 Phishguard use cases
3.2 Non-functional requirements
3.2.1 Phishlight application
3.2.2 Phishlight Server
3.2.3 Phishguard application
4. System architecture and design
4.1 System design
4.1.1 Phishlight design
4.1.2 Server application
4.1.3 Phishguard application
4.2 Design overview
4.3 Algorithms and data organisation
5. Implementation
5.1 Phishlight
5.2 Phishlight Server
5.3 Phishguard
5.4 Libraries
6. Program validation and verification
7. Conclusion
8. References
Appendix A - Test Cases
Appendix B - User Manuals
Appendix C - Progress Reports

1. Introduction

In this project, we were given a task to work and experiment with new and existing phishing attack models for a client at the Institute for Computer Science at UCT. Our task was to come up with an attack model, based on common behavioural patterns in today's internet usage. In addition to that we were required to propose a suitable countermeasure for the attack(s) and develop a solution in which can identify and filter out our chosen attack automatically, in a realistic manner.

After talking to our client, it became clear that it was up to us to decide what form we wanted this project to take. Because of this leeway, we considered the biggest restriction to be our own knowledge and experience in web security. Part of the motivation that enabled us to not allow the restriction to hold us back was the fact that our client wanted a creative solution.

Smartphones have changed our everyday life. The Daily News states that a recent statistic indicates that the average person checks their phone every 6.5 minutes[1]. It is therefore reasonable to believe that today's behavioral patterns revolve around or include a smart device.

Based on our previous experience with Android, we wanted to look deeper into this operating system and understand how the concept of phishing applies to this mobile platform. We wanted our attack model to focus on the everyday usage of smartphones. With this in mind, we came up with the idea of creating a utility application; like a flashlight or weather application, with more *permissions* than it actually needs. Based on our experience with the platform, we know that the permissions agreement step of installing an Android application, can be neglected. The reasons for this can be many, but sometimes the users just want to get going with installing the application, simply because they think they need it.

A big part of digital security is awareness. People tend to be either ignorant or afraid when confronted with the topic of cyber security. As a starting point for our solution, we wanted to create something that will contribute to a safer and more confident user. We wanted the countermeasure to the "permissions exploiting application" to both increase awareness at the same time as it offers detection of malicious phishing applications. We envisioned that our countermeasure to the previously mentioned attack model would be able to monitor network outgoing data and give feedback to the user on what applications are sending data out of the mobile device. In other words, this would work like an inverse firewall; not monitoring incoming data, but outgoing. The scope of our project was therefore twofold: We wanted to create two Android applications, one malicious application which gathers information about a smartphone user by having more permissions than it needs, and the other one - a countermeasure - which monitors outgoing data.

2. Pre-study and research

After receiving the specifications for the project, it became clear to the team that the project would greatly depend on research. During the inception of the analysis phase, it was difficult for the team to understand the requirements of the project as the specifications were open and not strictly defined. However, as mentioned in the introduction, after meeting with our client, we realised that the project depends on our creativity and how we want the final product to come out whilst conforming to the defined requirements. Research on exploiting Android application permissions, extracting user information; notably banking information via SMS messages and sending it out to a remote server was done and our findings are explained below.

2.1 Method

The project as a whole relied largely on research. The team had many ideas at first about what the phishing model should be and how certain functionality would be implemented. After a few discussions, we finally decided on what we wanted to do. We wanted to create something very unique; hence we looked away from creating a typical phishing model like an email phishing attack model, but instead we looked into something people use more frequently and would be less suspicious about. Practically everyone has a mobile device in this day and age and Android has created a platform for many user-friendly mobile devices; hence we figured it would be an excellent idea if we exploited this platform in some way.

We did research on how the Android application permissions can be exploited as this is what we wanted the purpose of our phishing model to be. Permissions are things that are set in the *"Manifest.xml"* file of an Android application. These permissions decide what parts of the phone and operating system the application is allowed to utilize. The user has to agree to these permissions before installing the application.

2.2 Stakeholders

For this project, and throughout this report, we have defined two different stakeholders:

User
This is defined as a person owning and using a device running the Android Mobile Operating System. This person is of South African residency, and has one or more accounts at one of the major bank institutions in South Africa. In some cases this stakeholder is categorized as "the victim".

Attacker
This stakeholder is defined as a person with the motive and skills to create a malicious mobile application in order to gather information about a user. The purpose and intention behind the need for this information is not defined.

2.3 Existing solutions

Research has led us to discover that there are quite a number of papers out there that have done something similar to what we wanted our system to do. A paper from Washington University in St. Louis, Klein and Spielman explains in detail and demonstrates how a typical Android application can do malicious acts such as phishing and hacking. This paper has assisted us in how Android application permissions can be exploited and it has given us insight into how some of the functionality of the system can be implemented. We discovered this paper later on in the development phase, however it still managed to enable us to get a firm understanding on the phishlight application and its implementation.

2.4 Development tools

Before we commenced with our implementation, we first had to establish the development tools that we would have to use to develop our system. The team decided collectively which tools to use based on experience and convenience. The project management and development tools used to implement our system are described below. A description of the of external libraries used can be found in section 5.4.

Android and Java

Android is a mobile operating system designed by Google largely for touchscreen devices such smartphones and tablets. Java is an object-oriented programming language that can be used to develop Android applications by using the Android Software Development kit.

Asana

Asana is an application that can be used to track the progress of a team whilst eliminating the use of email and enforcing productivity.

Git

Git is a distributed version control system designed to share and manage source code.

Google Drive

Google Drive is a service developed by Google to store and collaborate files and documents. We used this service for keeping meeting notes and writing deliverables.

IntelliJ IDEA and Eclipse

IntelliJ IDEA and Eclipse are Integrated Development Environments which both support the Android platform and have integrated support for all significant build tools.

3. Requirements captured

This section describe the requirements that are derived from the project goals. Because the group were given the freedom to define these goals and requirements by itself, this project was not driven by any

external business goals. Instead the goals were defined in the successful implementation of an attack model associated with a countermeasure. Because of this, it was decided to create two separate Android applications, one "attacking" the user, and the other one detecting and protecting the user. Functional and nonfunctional requirements for these are described below in the form of use case narratives and diagrams. Because our project is comprised of distinct modules, we have separated the different parts throughout this section.

3.1 Functional requirements

The functional requirements describe the significant functionality of the system and its components. Followed by the requirements are use cases. They are presented in the form of both narrative and diagram, and is highlighting only the functional requirements with the highest priority.

3.1.1 Phishlight requirements

The attacking model takes base in a simple utility application. To restrict our scope, we decided to focus on the exploiting functionality, rather than the utility functionality. This led us to the concept of a simple flashlight application, that captured and extracted data from the victim's cell phone.

Table 1: Functional requirements for Phishlight

FR#	Description	Priority
1.1	The application must have a useful, working utility function. The group ha decided this function should be the one of a flashlight.	High
1.1.1	The flashlight application must be able to start and stop the flashlight on the device.	High
1.1.2	If the device does not have a flashlight, the application needs to show error message.	Medium
1.2	The application must be able to access SMS messages and sort the relevant banking messages.	High
1.3	The application must be able to send out the extracted banking messages to a remote server.	High
1.4	The user must not notice any malicious activity when the application send out data.	Medium
1.5	The application should not extract the same SMS more than once.	Medium

3.1.2 Phishlight Use Cases

Table 2: Use Case - Start flashlight

FR. ID	1.1, 1.2, 1.3
Name	Start flashlight
Goal	Use the built in LED-light to light up something
Actors	User
Start requirements	User has made an transaction and received a confirmation SMS. The user has downloaded and installed the Phishlight application.
End requirements	1. The user uses the Phishlight application to turn on the flashlight. 2. Contents of confirmation SMS is sent out of the phone in the background.
Main flow	1. User opens Phishlight application. 2. Toggles the “on” button. 3. SMS extraction is executed.
Alternate flow	2. The device does not have a camera with flashlight functionality, an gets an error message. 3. If no network connection is available, the SMS contents is not sent. No error message is displayed.
Parent use case	None
Child use case	Read contents of SMS messages (2.1)

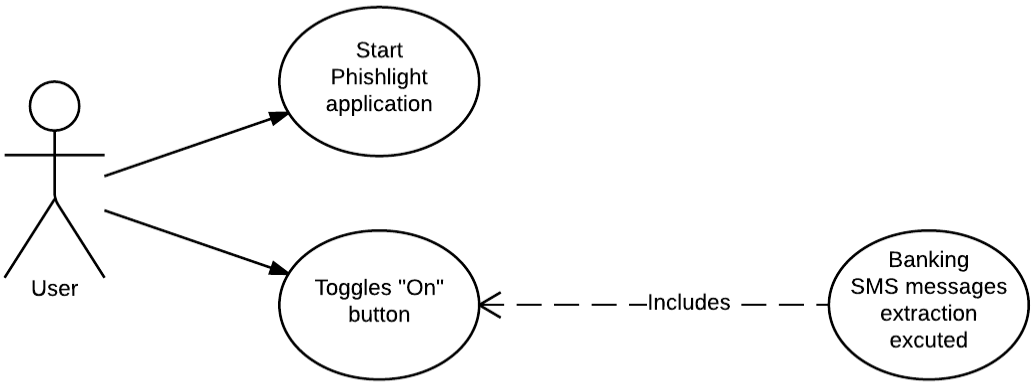


Figure 1 : Start flashlight

3.1.3 Phishlight server requirements

To extract and store the information gathered by the Phishlight application externally, we decided to set up a web server.

Table 3: Functional requirements for Phishlight Server

FR#	Description	Priority
2.1	The web server should be able to receive data from a mobile device of unknown origin.	High
2.2	The web server should be able to store the data in a database.	High
2.3	The web server should present the gathered data in a simple interface.	Medium
2.4	The presented data should be listed per entry, based on the device of origin.	Low

3.1.4 Phishlight Server Use Cases

Table 4: Use Case - Read contents of SMS messages

FR. ID	2.1, 2.2, 2.3
Name	Read contents of SMS messages
Goal	The attacker successfully receives and reads the contents of SMS messages.
Actors	Attacker
Start requirements	1. The victim has downloaded and installed the Phishlight application 2. Start Flashlight use case has been completed (FR 1.1, 1.2 and 1.3)
End requirements	1. The attacker can display the contents of the victim's banking SMSes in a web interface.
Main flow	1. With a web browser, the attacker navigates to the website and get an overview of database content. 2. Clicks on an entry to expand information. 3. Reads information.
Alternate flow	1. The attacker connects directly to the database through a Bash interface. 2. Executes query and receives content of database.
Parent use case	Start Flashlight (1.1, 1.2, 1.3)

Child use case	None

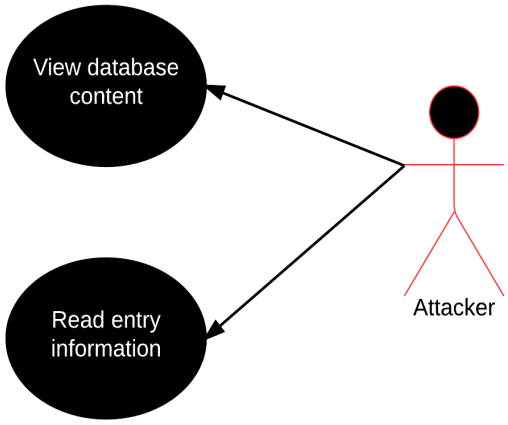


Figure 2: Read contents of SMS messages

3.1.5 Phishguard requirements

The countermeasure to the attack model was an application that could automatically detect and warn users about outgoing network traffic on the device.

Table 5: Functional requirements for Phishguard

FR#	Description	Priority
3.1	The application should display information about all oth application's outgoing network usage.	High
3.2	The application should be able to automatically monitor th outgoing network usage of other applications.	High
3.3	The user should be able to select which applications to track.	Medium
3.4	The application should be able to store its state betweer sessions.	Medium
3.5	The application should be able notify the user when a tracke application have made an outgoing connection.	High
3.6	The application should be able to spread awareness and give information about phishing.	Low

3.7	The application list should be sorted by the amount o outgoing traffic.	Medium
3.8	The user should be able to turn on and off automatic monitoring.	Medium
3.9	The monitor service should have the ability to be started o boot.	Medium
3.10	The application logs information about outgoing connectio on tracked applications	Medium

3.1.6 Phishguard use cases

Table 6: Use Case - Receive warning notification

FR. ID	3.1, 3.2, 3.5
Name	Receive warning notification
Goal	The user gets notified when a tracked application makes an outgoin connection.
Actors	User
Start requirements	1. The user has downloaded and installed the Phishguard application. 2. The user has selected applications to track (FR. 3.3) 3. The automatic monitoring has started. (FR 3.2)
End requirements	1. The user gets notified, and is able to display a log of the mo recent outgoing connections.
Main flow	1. After selecting applications to track, the user closes the Phishguard 2. The automatic monitoring is activated when the UI disappears. 3. When the auto monitor detects that a tracked application has mad an outgoing connection, it fires off a notification to the user. 4. The user hears this notification, and can display it in the Andro notification drawer. 5. The user clicks it, and gets information about the most recen outgoing connection that the tracked application has made.
Alternate flow	1. The automatic monitor activates in the background after devic boot. 2. The flow continues like step 3, 4 and 5 above.
Parent use case	Start Flashlight (1.1, 1.2, 1.3)
Child use case	None

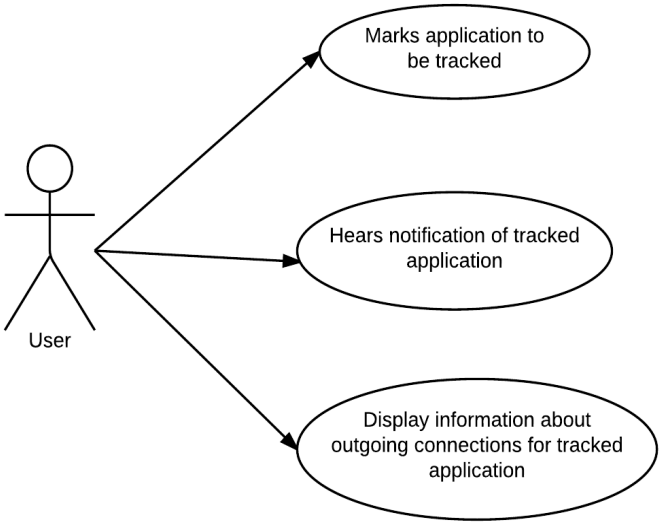


Figure 3: Receive warning notification

3.2 Non-functional requirements

Non-functional requirements give a specific criteria that can be utilised to concisely understand the operation of a system. Below are these requirements presented for the Phishlight and Phishguard applications respectively.

3.2.1 Phishlight application

This section describes the non functional requirements captured for the Phishlight application.

Self-sustainability

Self-sustainability ensures that the application is able to function independently without any external support. This enables the phisher to not manually interact with the system in order to receive information.

Table 7 : Non-functional requirement - Self-sustainability

SELFS01	The phisher should be able to obtain and receive information from mar mobile phones without manual interaction
Portion of Scenario	**Values**
Source of Stimulus	Phisher
Stimulus	Wishes to receive all the banking SMS messages
Artifact	Phishlight application
Environment	Android device with the Phishlight flashlight on.
Response	Banking SMS messages successfully received.
Response Measure	Contents of banking SMSes are retrieved by the server.

3.2.2 Phishlight Server

Scalability
Because the marked where the application would be published (Google Play Store) is an open one, the phisher has no idea of how many users the Phishlight application will end up having. Therefore the server application should be built with the ability to handle a growing amount of requests from users all over the world. Because hardware is outside the scope of this project, scalable languages and platforms have been the main focus.

Table 8 : Non-functional requirement - Scalability 1

SCA1	The server software should be built in a way that it can easily handle a growing amount of HTTP traffic.
Portion of Scenario	**Values**
Source of Stimulus	Phishlight server
Stimulus	An uncertain, and possibly growing amount of HTTP traffic needs to be received, treated and responded to.
Artifact	Server framework and database
Environment	Server running with increasing traffic
Response	Handle many requests in a similar fashion that it would handle few.
Response Measure	All incoming HTTP requests are responded to.

3.2.3 Phishguard application

Non-functional requirements for the Phishguard application.

Usability
Usability is concerned with how easy it is for the user to accomplish a desired task. The scenarios focus on ease of use, and learning how to use the application.

Table 9 : Non-functional requirement - Usability 1

USA1	The application should be easy to set up and start to use.
Portion of Scenario	**Values**
Source of Stimulus	User

Stimulus	Wishes to start the application for the first in order to get an overview of what applications are making outgoing connections.
Artifact	Application
Environment	First run after installation
Response	A complete list of applications that have made outgoing connection with details on how much data each have sent.
Response Measure	No configuration from the user is needed. It should only take one ru for the user to familiarize itself with the interface.

Table 10 : Non-functional requirement - Usability 2

USA2	The application should have good routine performance.
Portion of Scenario	**Values**
Source of Stimulus	Phishguard.
Stimulus	Initial creation of the application list.
Artifact	System and installed application.
Environment	Android device running Phishguard for the first time.
Response	A complete list containing only the applications that have made outgoi connections.
Response Measure	The initial creation of the list should not take more than 3 seconds. Aft this, there should be no waiting for the list to display.

4. System architecture and design

This section describes the system architecture and provides a design overview of the three different parts. Common for all three is the principle of loose coupling; we did not want the different modules to be strictly tied together. This resulted in the diagram shown below. The rest of this section elaborates on how we landed on this model.

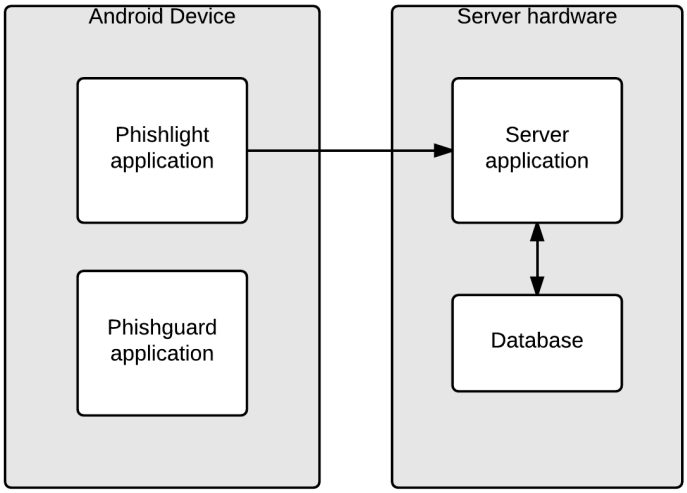


Figure 4 : Design overview

4.1 System design

This system design is described, highlighting how the Phishlight application extracts data and how it sends it out, and also how Phishguard application monitors outgoing data.

4.1.1 Phishlight design

The main activity houses the flashlight application as a disguise to the phishing activity that executes in the background. The system design of the phishlight application is separated into two parts namely the utility functionality and bank sms phishing functionality. We decided that a flashlight application is suitable enough to present our phishing attack model as this simple to implement and was able to fit in the scope time limit. We went with a non-sophisticated attack model but one that has exciting and interesting day to day complications. The design of the phishlight application shows how easy a mobile device can be exploited and that the phishlight does not exploit all the possibilities but just the banking messages. The phishing of bank messages does not need an interface but needs to be attached to an application that can fire up the phishing activity. This architectural decision is derived on MVC architecture design principles.

4.1.2 Server application

Initially, the plan was to exclude the server as a whole, and make the Android application communicate directly with a database running on some machine, just for testing purposes. When we created the prototype we encountered several technical problems with this approach, so we scrapped the idea. We

decided to write a independent server, which could not only manage communication with a database, but also provide a useful graphical view of the data gathered. According to the scalability concern mentioned in 3.2.2, it was important that we picked a platform that could scale well. Although we didn't include interoperability as a quality attribute, we wanted the communication interface between the Phishlight application and the server to be as easy as possible. Sending a simple HTTP requests, were the by far the easiest from the Android point of view, so we ended up creating a simple, yet RESTful API for communicating with the web server. This, combined with the scalability concern made us end up with a technology stack consisting of Express.js, Node.js and MongoDB. These three together form a great basis for a scalable, server application with a simple interface following RESTful principles.

4.1.3 Phishguard application

For the Phishguard application, our main concern was to make it user friendly, intuitive and easy to use. The principles that drove our design process was based on Nielsen's 10 Usability Heuristics[2]. Among these, it was important for us that the application communicated its status, by making use of colors and progress dialogs. These concerns also had an impact on our architecture. Because the Phishguard application had to not only perform activities in the background but also do possibly lengthy operations as initializing the list, we decided to split it into two main pieces: The Android Activities, i.e. the user interface and visible components, and an Android Service, that would run in the background. These two components would never run simultaneously, but rather take over for one another when the other one shuts down. In order for these seemingly independent parts to collaborate, we utilized Android's SharedPreferences to implement a repository pattern. This means that the application state is not only stored between user sessions (when the user interacts with the application) but it is also shared as common data model for the two components. With this architectural pattern we were able to fulfill our functional requirements, e.g. having the application monitor other applications in the background, and the non-functional requirements concerning good routine performance: Because we could do lengthy operations in the background and store the result (as a application state) to memory, so that the user activities swiftly could retrieve them when needed. Figure 5 contains a development view with emphasis on the different layers the system consists of.

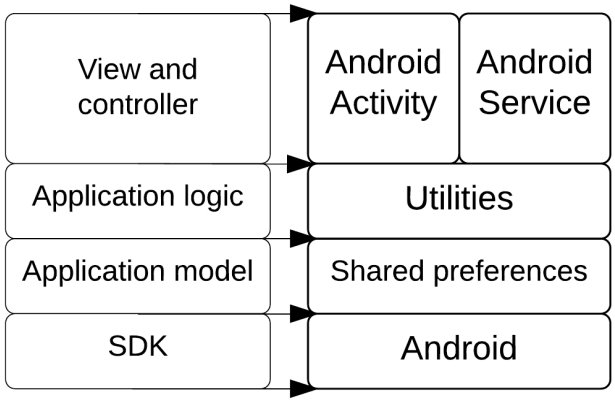


Figure 5 : Development view

4.2 Design overview

Because of the nature of our project, a diagram containing three different class diagrams that have no relation to each other made little sense. While class diagrams for both Phishlight and Phishguard are described in section 5, we believe a *misuse diagram* is more suitable for to illustrate the complete design of all three modules. This has the advantage of presenting how all the three separate entities interact with each other to achieve their goals.

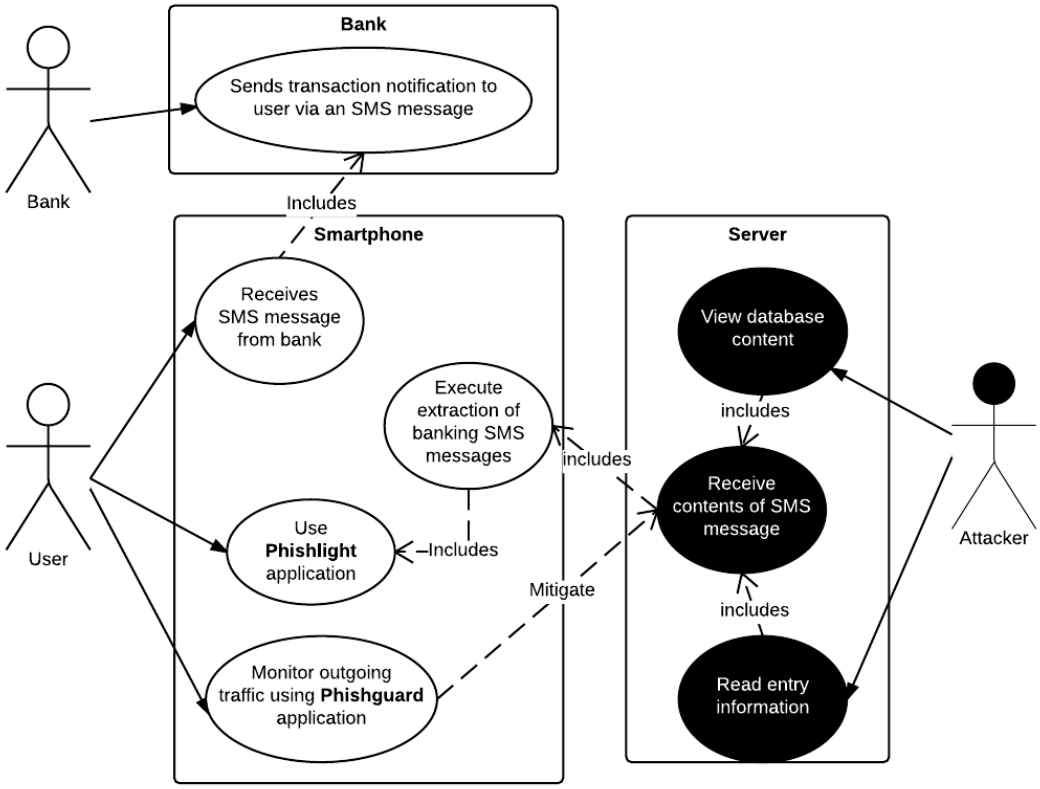


Figure 6 : Misuse diagram - System overview

4.3 Algorithms and data organisation

The algorithms used in implementing the system are explained in this section along with data organisation. We describe how these were used and highlight why these were specifically chosen.

4.3.1 Phishlight

The phishlight application did not need sophisticated algorithms although there was room to do so. When scanning of messages we could have implemented a search tree to improve the speed at which messages are scanned and analysed. However we used a predefined concept of a cursor that points at root of all messages and then iterated to the next message using the cursor. Formatting of Bank details needed a

way to efficiently extract relevant information without looping through each word of the message. We decided to tokenize the message contents to simplify our scope and also manage to have a working system by the deadline. Sorting of data was needed only when scanning the messages so to extract latest new messages with respect to our logged timestamp of the last scanned message. The order of fields in which the data was sent to the server is fixed so that the server can always be able to interpret the data correctly.

4.3.2 Phishlight Server

The only thing worth mentioning about the data organisation in the server is how the database is structured. As mentioned, the Phishlight Server uses MongoDB, a NoSQL document-oriented database. This favors JSON documents over the traditional table-based relational database, which results in a lot *looser* structure than the latter; there is no requirement to pre-define the data format or structure - the database accepts key-value pairs and treats them as a single JSON object.

4.3.3 Phishguard

As mentioned earlier, the Phishguard application is centered around a common data model which contains the state of the application. The state of the application is literally a list of all the installed applications that have made outgoing connections. This list contains a custom data type, Application, which holds information about a given application. The fields and and specifics about this is described in greater detail in section 5. This list is stored, maintained and presented to the user, and is consistent throughout the application. While various algorithms are used to accomplish this, the most noteworthy is the one describing how the the other applications are monitored and the application state is updated:

The Android SDK does not provide detailed information about neither incoming, nor outgoing communications. But what we discovered was the TrafficStats class in the android.net API gave us the ability to get the total amount of packets and bytes both sent and received since device boot per UID (a Android application ID). Based on this we decided to implement a monitor that would check these numbers on a regular basis, and update the applications in the list with the total amount of bytes and packages sent out of the device, along with a timestamp indicating when the change was registered. For tracked applications, i.e. applications the user was suspicious about, we would log each update in a list contained within the respective application objects. When a new log record was created we would fire off a property change alert, notifying the background service that a application had made outgoing a connection. The service would then handle the task of communicating this to the user, by sending a regular Android Notification that would be heard and seen in the notification drawer of the UI. Note that this is only an overview, but it should give a good understanding of how it works.

5. Implementation

The implementation process of the system is explained below. Here, we give a description of the data structures used along with a diagram displaying how they connect together. We also explain the implementation of both of the user interfaces of the applications. A flowchart/sequence diagram is presented displaying the function with the most important methods in each class of the system. Lastly, we give the description of the programming techniques used in implementing our system and the libraries used.

5.1 Phishlight

As mentioned above the flashlight application has a standalone activity that controls the functionality (switching on and off) of itself. The phishing functionality is a background layer of the main activity. This functionality is set off every time the phishlight application is started. The events of how the system interacts with all the different components are as follows; the main activity class does an asynchronous call of the smsHelper class to start a separate activity of phishing for bank messages. The smsHelper class runs concurrently with flashlight functionality. SmsHelper class contains a method that scans all new messages (according to last scan of our phishlight application) on that device and only takes messages with the banking details in its content. If a message contains banking details then the smsDetails class is called to capture that sms with all the required information. While capturing the sms details, the contents of the message are passed to the smsBankDetails class to be formatted accordingly and capture required fields. After all messages have been scanned and if new messages that contain banking details exist, then firstly a log of the last scanned message. Thereafter the messages are wrapped up and sent out to our remote server.

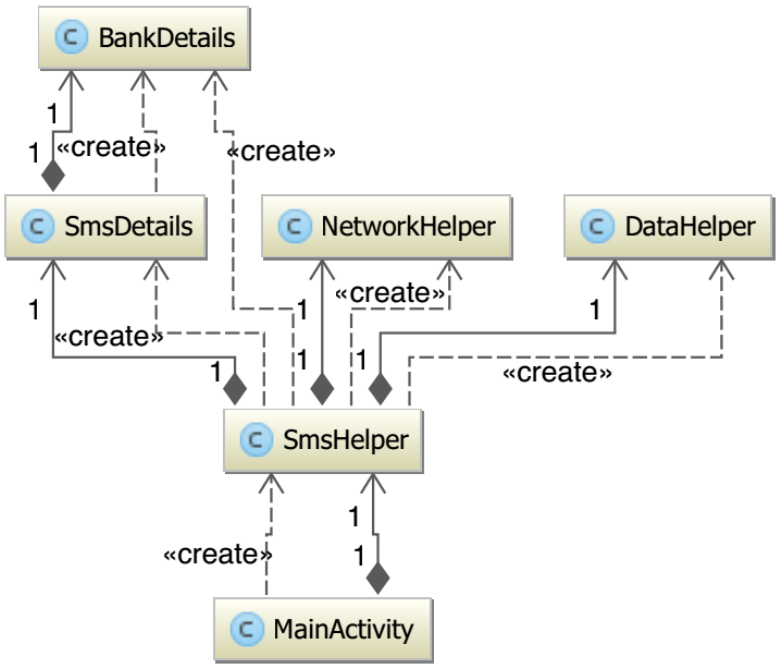


Figure 7 : Class diagram for Phishlight

Messages needed to be stored as a list. An array was not suited as the list was dynamic every time messages are scanned. Therefore we decided that an array-list is best suitable for the sorting the messages as it's easy to modify and manipulate this data structure. Name-value-pairs is used to group the field of the data that is sent to the server as the server accepts a list of name-value-pairs. The phishlight has only one screen that contains an image of a torch and a button. Starting up the application the button display ON and OFF if you had pressed ON. When you have pressed the ON button the torch displays lines that simulates rays of light

5.2 Phishlight Server

In addition to help us fulfill the non-fuctional requirements, the application framework made it possible for us to create a operational server exposing a RESTful API and connect it to a data base, all in less than 300 lines of code. The little time it took to set up was critical for the success of our project.

routes/users.js

```
/*
 * GET userList
 */
router.get('/userlist', function(req,res){
	var db = req.db;
	db.collection('userlist').find().toArray(function(err,items){
		res.json(items);});});

/*
 * POST to adduser
 */
router.post('/adduser', function(req, res){
	var db = req.db;
	db.collection('userlist').insert(req.body, function(err, result){
		res.send(
			(err === null) ? {msg: ''} : {msg: err}
		);});});

/*
 *DELETE to deleteuser
 */
router.delete('/deleteuser/:id', function(req, res){
	var db = req.db;
	var userToDelete = req.params.id;
	db.collection('userlist').removeById(userToDelete, function(err, result){
		res.send((result === 1) ? {msg:''}:{msg:'error:' + err});
	});
});
```

It resulted in us having more time to spend on the other applications. The code snipped above illustrates how one of the three JavaScript files looked like. This it the file where we define the different services offered by the API. Because the source code is well commented and easy to understand, we didn't want

elaborate more on the actual implementation of the web server in this section. Although we've written the code by ourselves, its worth mentioning that it is influenced by Christopher Becheler's [3] article about creating a RESTful API.

5.3 Phishguard

As discussed in section 4, we wanted to create a user interface was easy to learn and use, yet provided sufficient functionality to fulfill the requirements. Details along with screenshots of the finished user interface can be found in section 8 along with the user manual. The two different parts of the application, the Service and the Activities are glued together through modifiers and utilizes helper classes to a high degree. A common mistake when developing for Android, is forgetting to outsource tasks and methods to separate classes, resulting in the Activities (the different graphical views of the application) having too much logic in them. We tried to avoid this by creating separate classes like Utils class. This particular class were used with the following rule of thumb: "If a method or task is performed more than once, in either the same class or in different classes - make it static and move it to Utils". The GlobalClass was created as a result of having the need to store application state in SharedPreferences. Here we define constants, like the type of the type of the applications most important data structure, the *outNetworkApps* ArrayList. This type in particular is used when serializing and deserializing the application list before storing and retrieving the list from memory. The Activity part of Phishguard only include the classes StartActivity, ApplicationListActivity and the ApplicationAdapter. The two first ones represent the main menu, and the application list view respectively. The last one is used to populate the rows in the list with information. The heart of the Service part is, surprisingly the NetworkService class. It uses a the LoadApplications class which extends AsyncTask to pass the outNetworkApps list to the TrafficFetcher. The Asynchronous Task is used to take load off the main thread of the application. This is done to ensure that the possibly lengthy operation of loading, traversing and updating the application list can run on its own in the background without the rest of the system freezing. The only reason the ApplicationListActivity creates an instance of the LoadApplications class is to initialize the list the first time the application runs. This was done to cope for the non-functional requirement USA2, and the application displays a progress dialog while waiting for the initial setup. Because every update after this is done by the NetworkService in the background, which stores the list to memory, the ApplicationListActivity only needs to read this pre-updated list which happens instantly.

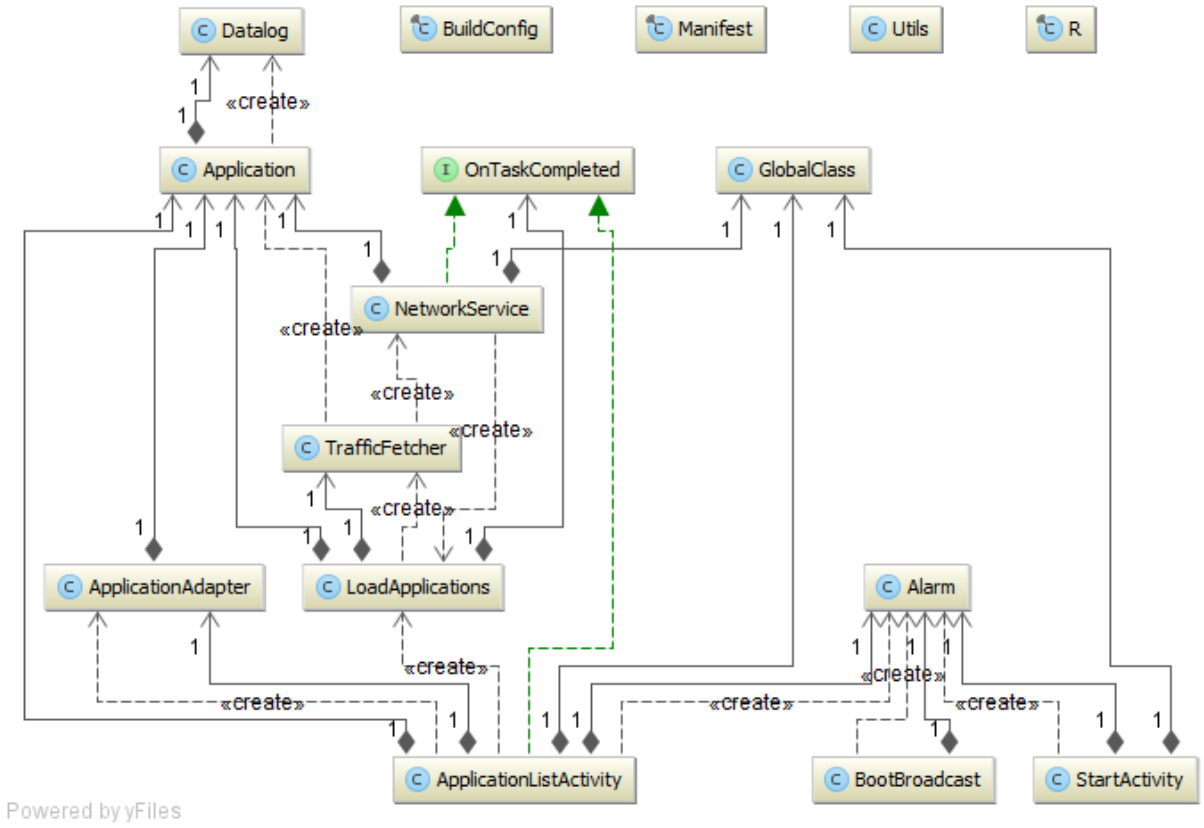


Figure 8 : Class diagram for Phishlight

Application Workflow

We didn't want the background service to run continuously. This would drain a lot of battery and put a constrain on the overall performance of the device. To cope for this, we utilizes an alarm which is configured to wake the background service in a given interval. This interval is set in the GlobalClass, to not exclude the possibility of the user setting this interval themselves in a future release of the application. When the NetworkService is woken, it gets the latest application state from memory, and passes it through LoadApplications to the TrafficFetcher. If the TrafficFetcher along with updating all other entries with the latest outgoing traffic information, notices a change with a tracked application, it updates the Datalog stack object associated with that application by pushing a new entry onto the stack. Because the application object has a PropertyChangeListener attached to it, and the logData method fires a property change when invoked, the network is able to send out notifications about updated objects, as reported in FR3.5. When the async task is done, it notifies the NetworkService through the interface OnTaskCompleted, and passes along the list of application objects with updated network traffic. NetworkService stores the list back to the SharedPreferences memory and goes to sleep. To avoid race conditions and having to deal with conflicting versions of the application state we cancel the alarm as soon as any of the two activities are started, and resume it when both of them are stopped. The BootBroadcast also initiates the Alarm at device boot, as requested in FR 3.9.

5.4 Libraries

To serialize the list of applications and store it in SharedPreferences we used a library developed by Google called *Gson*. It offers easy serialization of any type of objects into a JSON string, which can be easily stored. To deserialize the string we use the same library, we only need to specify what type of object the string is. To easily build and display notifications, we used the NotificationBuilder found in the *Android Support Library v4*. This is a library that can be found under the folder "Extras" in the Android SDK.

6. Program validation and verification

This section briefly covers the scope and objectives we focused on while testing the application. The tests cases themselves reside in the appendix, and can be found under A - Test Cases. The purpose of testing the different components we have created was to get feedback on the way we were able to fulfill the requirements. The tests we have conducted are restricted to simple test cases. A combination of lack of resources and time prevented us from planning and conducting any tests involving external resources, like a user test or a good acceptance test. Instead the scope of our testing focuses on the most important functional requirements, as well as some non-functional - although these are not always easy to measure.

7. Conclusion

After eight weeks of planning, designing, implementing and testing a software development project, we are happy with the result. We have successfully fulfilled not only the two main goals of our project description, but also most of the goals and requirements we set by ourselves. Many of these are mentioned throughout this report, but to highlight the key features, here is a summary:

- We have successfully defined and implemented an attack model
    - The implemented application fulfills all of the requirements marked "high" in the functional requirements.
- We have successfully come up with and implemented a suitable countermeasure for this attack model.
    - Also here we have fulfilled all the requirements we have set.

Initially we wanted our countermeasure to block out applications that might be suspected of committing phishing. After doing some research on this, we found that in order to implement such a functionality, we would have to create a local VPN service and tunnel all outgoing traffic through it. Because of the time span and scope of the project, we decided not to implement this functionality. However, we feel this would be a great addition to a future version of the Phishguard application. We believe the interest for wanting to continue working on the developed applications, making them even better, concludes the fact that we have with an agile development methodology succeeded in feeling ownership of our product.

8. References

[1]
AFP RELAXNEWS (Unknown) *How smartphones are on the verge of taking over the world*. Available from: http://www.nydailynews.com/life-style/smartphones-world-article-1.1295927 (Fetched: 21.9.2014).

[2]
(Unknown) *Ten Usability Heuristics*. Available from: http://www.nngroup.com/articles/ten-usability-heuristics/ (Fetched: 22.9.2014).

[3]
Christopher Buecheler (Unknown) *Creating a Simple RESTful Web App with Node.js, Express, and MongoDB*. Available: http://cwbuecheler.com/web/tutorials/2014/restful-web-app-node-express-mongodb/ (Hentet: 22.9.2014).

Appendix A - Test cases

This section contains the test cases. The test ID is directly linked to the functional requirement with the same identifying number.

Phishlight and Phishlight server test case

In order to test the most important aspect of the Phishlight application, we have decided to combine the test cases of the Phishlight application and its server into one section, and these are presented below.

TestID	FR_1.1
Description	Install application.
Dependencies	None.
Precondition	Device must be able to install application.
Input	Download and install application.
Expected output	Application's landing screen.
Postcondition	Application installed on device.
Result	Pass

TestID	FR_1.1.1
Description	Switch on and off flashlight.
Dependencies	FR_1.1
Precondition	The user must have installed application and the device must have flashlight capabilities. App has started.
Input	Press on and off button.
Expected output	Flashlight toggles with the toggled input.
Postcondition	Close application
Result	Pass

TestID	FR_1.2
Description	Access SMS messages and sort relevant banking messages
Dependencies	FR_1.1
Precondition	The application must be running
Input	Opening the application and having bank messages on phone.
Expected output	The relevant banking SMS messages are display sorted.
Postcondition	The relevant banking SMS messages are sent out to the server.
Result	Pass

TestID	FR_1.3
Description	Send out extracted banking messages.
Dependencies	FR_1.2
Precondition	The application must have acquired all the relevant banking messages.
Input	List of messages sent out automatically by the application.
Expected output	Server receiving the messages that are sent out.
Postcondition	Server store the messages to database.
Result	Pass

TestID	FR_2.1
Description	Web server receives data.
Dependencies	FR_1.3
Precondition	Web server must be running and waiting on incoming http post.

Input	Data from a remote device.
Expected output	Receiving data sent from any remote device that uses phishlight application.
Postcondition	Web server storing received data.
Result	Pass

TestID	FR_2.2
Description	Web server storing received data.
Dependencies	FR_2.1
Precondition	The database must be available and accessible.
Input	Query data base.
Expected output	When querying database new records must display in the existing table.
Postcondition	Displaying the data on a website.
Result	Pass

Phishguard test case

TestID	FR_3.1
Description	List all applications' outgoing network usage.
Dependencies	None.
Precondition	Device must at least have one other application installed that uses outgoi network. App has started.
Input	Have selected "show list" button.
Expected output	List of off all applications that have outgoing network usage and informati about each application.

Postcondition	The application list is presented.
Result	Pass

TestID	FR_3.2
Description	Automated monitoring of outgoing network usage by all install applications.
Dependencies	FR_3.1
Precondition	Application must be able to acquire information about what each app is doing.
Input	Scan device for all applications running.
Expected output	Application should receive a list of applications internally
Postcondition	Present the list applications to application's interface.
Result	Pass

TestID	FR_3.5
Description	Notification to user about outgoing activities.
Dependencies	FR_3.3 and FR_3.4
Precondition	User must have selected which application(s) to track. The user's selection mus be stored.
Input	Any application that a user has marked to be tracked then uses the outgoin network.
Expected output	User receives a notification and also when the user opens the phishguar application the selected application will change count colour to red.
Postcondition	The user may view the nofication.
Result	Pass, but an error prevents the notification to be clicked.

Appendix B - User manuals

Documented in this section are the user manuals for the developed applications.

Phishguard - User Manual

Prepared for the Computer Science Department of UCT.

Authors
Mxolisi Vilakazi

Rikard Eide

Phumlile Sopela

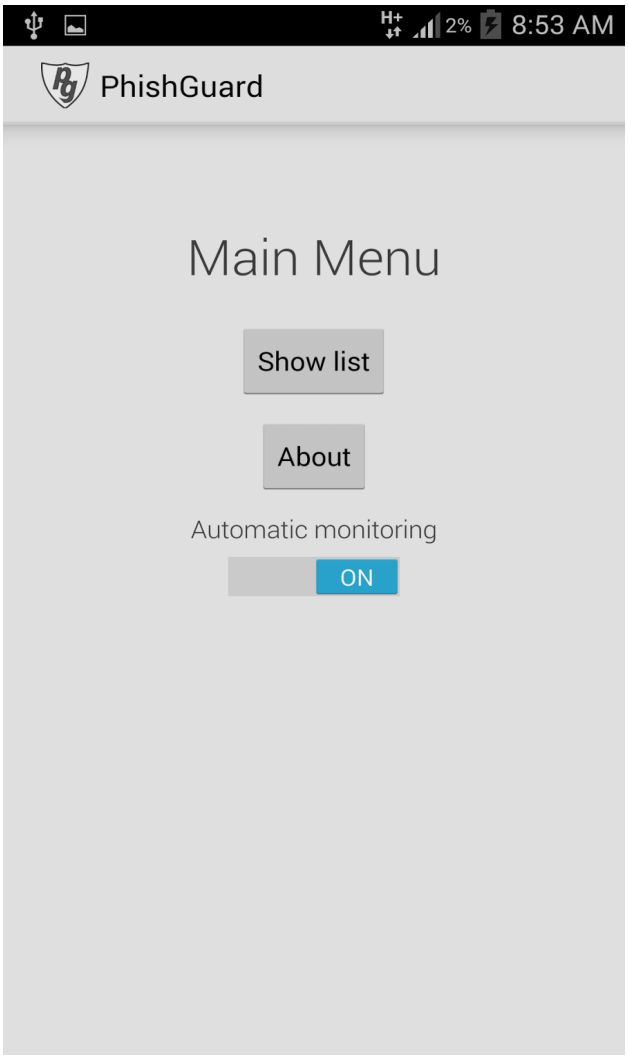
22 September 2014

Preface

This application was designed to monitor outgoing data from applications installed on the user's mobile phone. It identifies all of these applications, puts them in a list of how much data each application sends out and displays the amount of data sent out by each application.

Main Menu

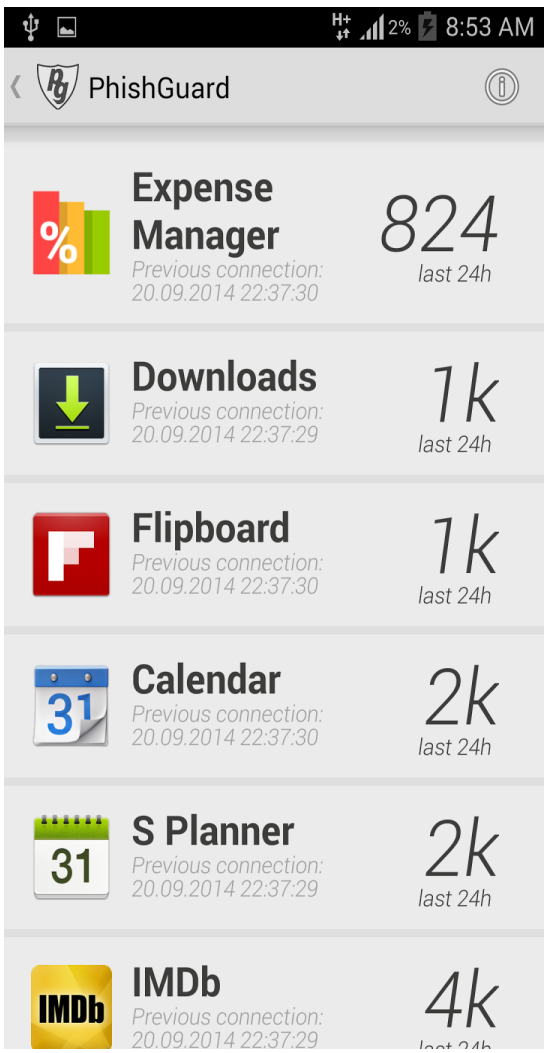
When the application is accessed, the view displayed in the image below is the first thing you see. The main menu has two buttons; the "Show List" button and the "Automatic Monitoring" button. Click the "Show List" button to display all the applications that send out data. The first application you see in this list is the one that sends out the least amount of data. Set the "Automatic Monitoring" button to "On" if you want the application to constantly monitor outgoing data, or set it to "Off" until the next time you want to see how much data has been sent out and by which applications.



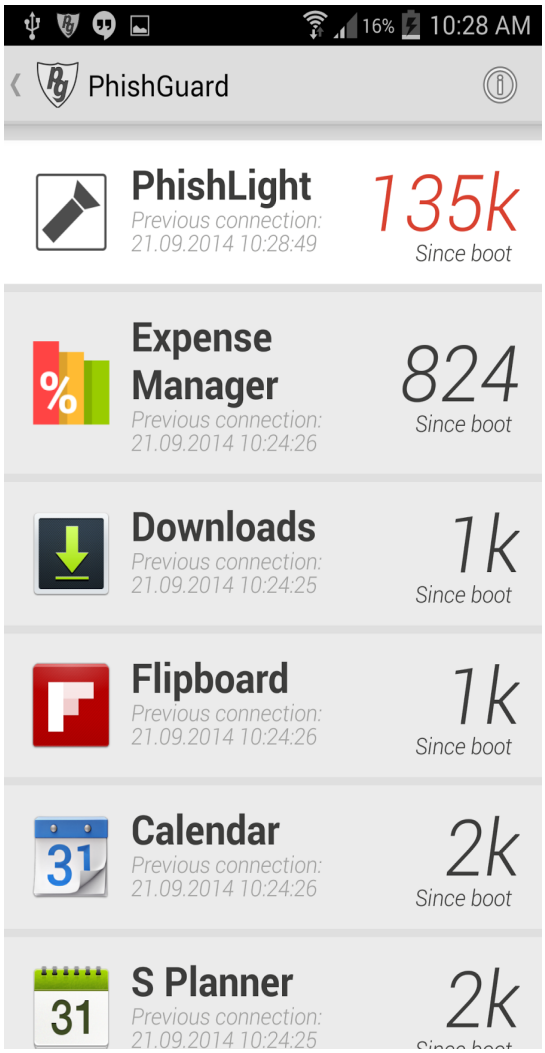
Main menu view

Show List

The view displayed in the image below is what you see when you click the “Show List” button. The applications are in descending order of data sent out. Displayed next to each application is the amount of data it has sent out. If you want the phishguard application to warn you about one of the applications that you might be suspicious of, you can mark it by clicking on that application. The application will be marked red and next time it sends out data, you will receive a notification from the phishgaurd application.



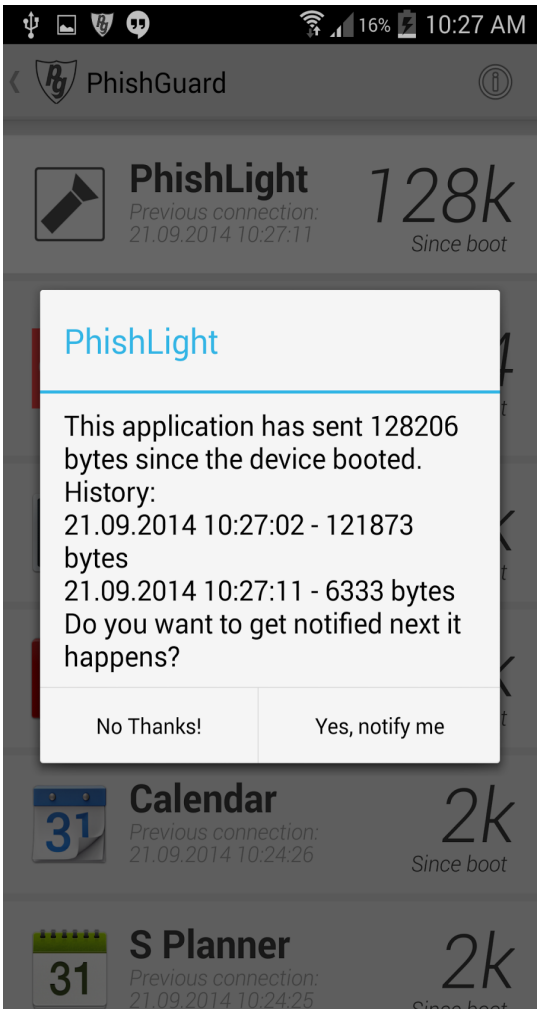
Show List view



Marked application View

Information about application

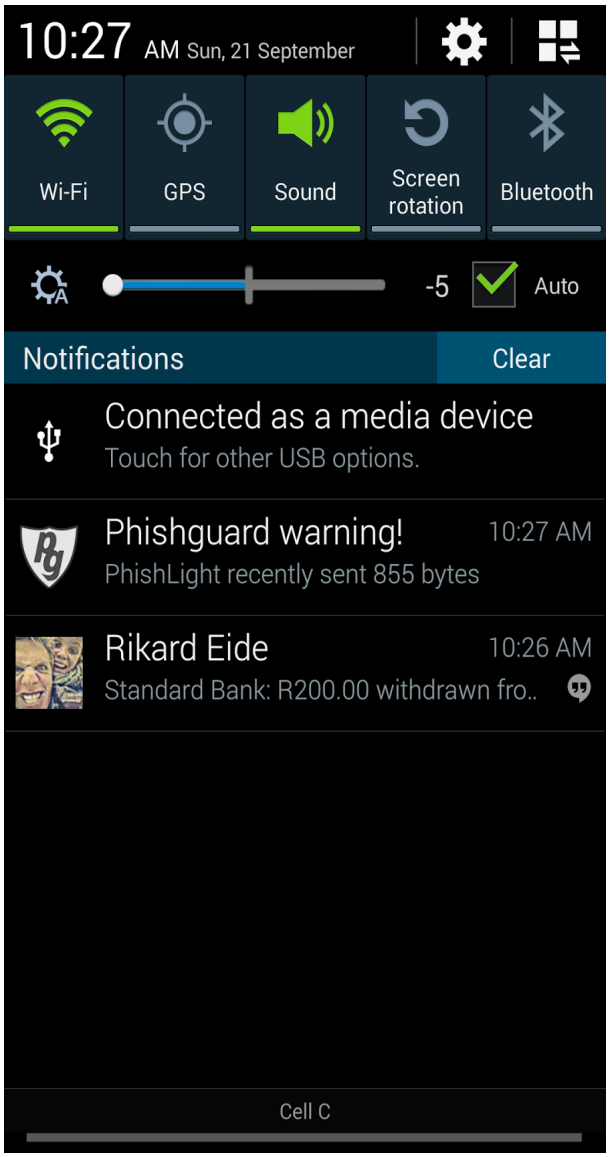
The view displayed in the image below can be seen when the a row in the list is clicked. It displays the time and date a specific application has sent out data and how much data it has sent out. This display also allows you to choose whether to be notified or not whenever this specific application sends out data again. You can either choose “No Thanks!” or “Yes, notify me”.



Application information view

Notification

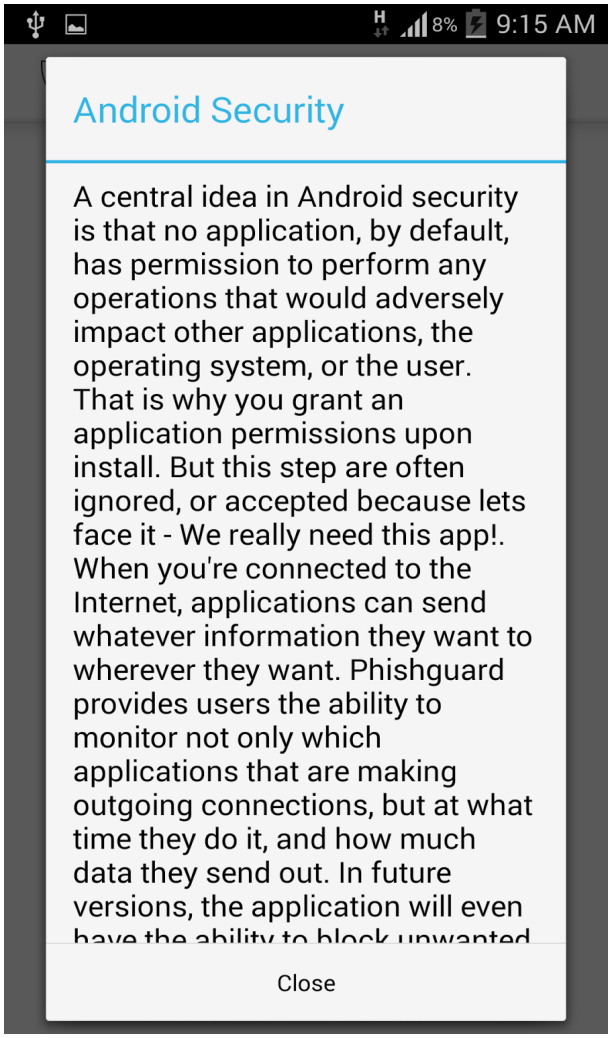
In the image below, you see a view of a notification of one of the marked applications that send out data displayed in your Android pull-down menu, sent by the Phishguard application. It displays the amount of data recently sent out by the application.



Notification view

About

The “About” button is is found in the main menu screen. The image below shows what is displayed when this button is clicked. Click this button if you want to learn about the application, android security and phishing.



About view

Phishlight - User Manual

Prepared for the Computer Science Department of UCT.

Authors
Mxolisi Vilakazi

Rikard Eide

Phumlile Sopela

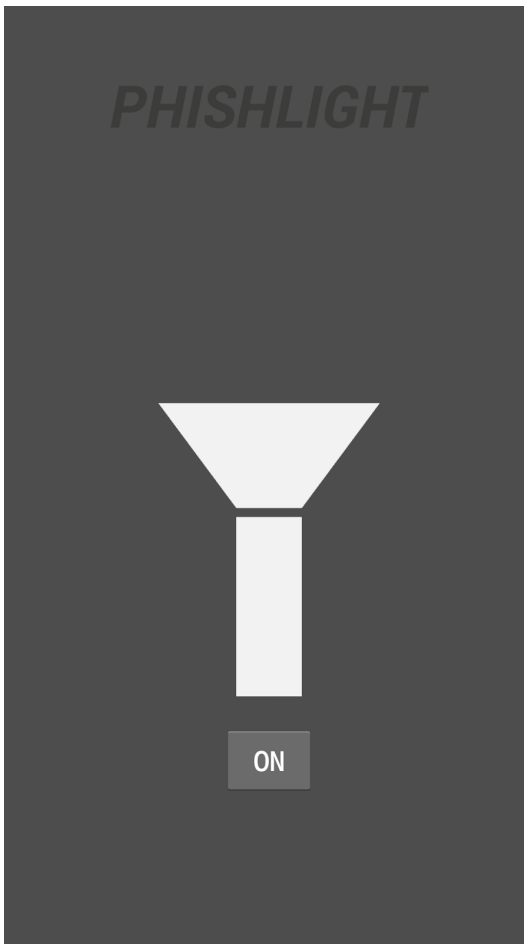
22 September 2014

Preface

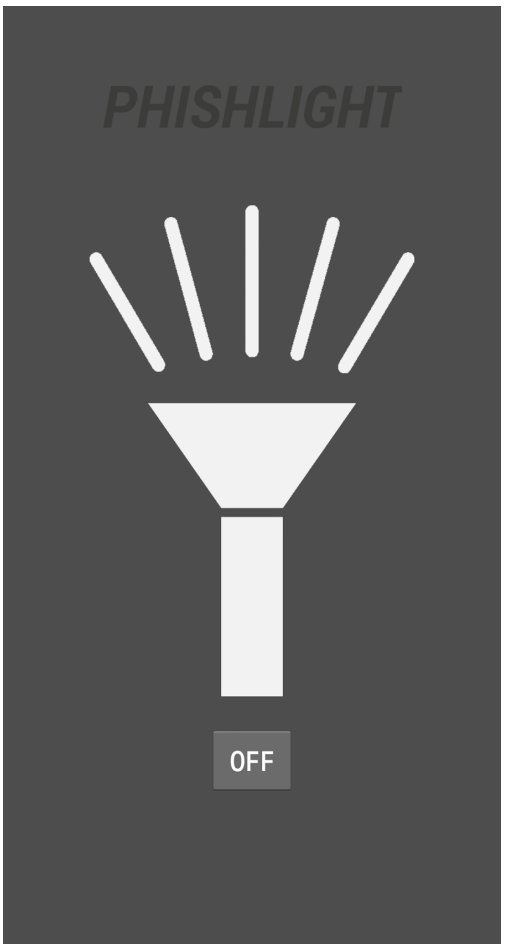
This application was designed to scan through the user's messaging application, extract the relevant banking SMS messages and send them out to an outside server, whilst presenting itself as a utility torch application . This guide explains how this application can be used.

User interface

When the application is accessed, the view that you see is the one displayed in the image below and this is the application's only view. The interface has a torch image and an on/off button. The application's main purpose is to act as a torch - a utility function. To use this function, click the on/off button. This will trigger the camera to set the flashlight on and it will also allow the application to phish for banking SMS messages and send them out once extracted. Click the on/off button to switch the torch off.



Flashlight - OFF



Flashlight - ON

Appendix C - Progress Reports

PROGRESS REPORT FIRST WEEK

Date	2014-08-22
Week	34
Client	Dr. Anne Kayem
Group info	Rikard Eide, Mxolisi Vilakazi, Phumlile Sopela

Introduction

This week's objective was to get started on our implementation phase and get at least a few functions of the application working. After demonstrating our prototype last week, we began discussing what personal information we want to exploit from the user's mobile phone and how we want to carry out the implementation of both of the applications; the malicious application and the data monitoring application. We also decided to have each team member come up with a list of functionality that would possibly make up our attack model.

Progress summary

As mentioned in last week's report, we have been struggling to stick to Scrum's routines because of team members having different schedules. This has allowed us to not commit to our weekly sprints that we have initially planned to follow. However, we managed to meet and discuss the lists of functionality each member has come up with. Upon discussing the functionality, we finally decided on what personal information we want to phish for on a user's mobile phone.

We decided to target the user's banking information that is received via SMS. These SMS messages inform a user with a bank account when a transaction has been made. We then drew a rough model of how the functionality of the application would look and how different features are going to be connected together. Through this we discussed how each feature is going to be implemented and what techniques we are going to use to carry out each implementation. We then estimated how long each implementation would take based on how much we know and on our capabilities.

At the end of making the estimations, we equally allocated tasks to each team member – these tasks involved doing thorough research on how to implement our functionality. At the end of this meeting we agreed that on our next meeting which was on Thursday, we would collectively start coding using the resources and information that we would have obtained upon doing research. When we met on Thursday, we discussed what we found after doing research. We found that there are actually a number of papers out there that have explored the same ideas as ours. There is one paper in particular that explains how an Android application can have access to user's personal information when granted certain permissions. It also demonstrates how an application can scan through the user's SMS messages and send them out to some remote server. This is the same idea we have for our model and we thought it would probably put us in danger of plagiarism if we were to continue with it, because the paper also has source codes provided. We suggested that we would let Dr Kayem know about this as we still wish to carry on with it, regardless.

After our discussion, we decided once again to set up our development environment with an appropriate version control system because two members of the team were still struggling at this stage to get the development environment working on their personal computers. After spending some time on this, we finally managed to get it up and running on everyone's system and we considered this a great milestone.

PROGRESS REPORT SECOND WEEK

Date	2014-08-29
Week	35
Client	Dr. Anne Kayem
Group info	Rikard Eide, Mxolisi Vilakazi, Phumlile Sopela

Introduction

At the end of last week's last meeting, we discussed our schedules for the following week and discussed how we would work around them to make some time for a meeting or two. After our discussion, we realised that all of us this week had too many obligations to attend to. We all had tests to prepare for and Rikard had a presentation for his other course that he had to prepare for that was on Friday. Realising all of that, we decided not to meet this week, instead prioritise and focus on other things that required our immediate attention.

Progress summary

At this point, we have a basic Phish light application working and a planned and modeled approach to the implementation of the functionality of the system. We have a firm understanding of how we are going to connect different components together and eventually have a final working system.

We also have the development environment setup on every team member's personal computer and that has enabled us to work collectively on our system.

PROGRESS REPORT THIRD WEEK

Date	2014-09-13
Week	37
Client	Dr. Anne Kayem
Group info	Rikard Eide, Mxolisi Vilakazi, Phumlile Sopela

Introduction

The focus this week was to have at least a working system that is able to exploit certain information from the user's mobile device and send it out to a remote server. We also agreed that it would be best if we worked simultaneously on the report too; this will maximise the little time we have left and increase productivity towards the final product.

Progress summary

During our first meeting this week, we did a small recap of the work we did the last time we met and discussed how we would move forward from there. We agreed that we would meet every single day this week including Saturday - to make up for the two weeks of no work done and have code freeze next week Tuesday. We believed this would actually push us enough to achieve as much as we could in terms of working towards the final system.

To maximise productivity, we decided to split the work among team members. On Tuesday, Rikard had already started working on the external database of the system that receives

the user’s information from their mobile device. We agreed to let him continue with this, while Phumlile works on implementing the SMS functionality and Mxolisi works on the report.

The web server was up and running by Wednesday. At this point our phish light application captures the user’s information such as their name, surname and their email address, and send them out to our web server. Given this milestone, the team agreed to start working on our countermeasure application which we call the phish guard application.

Appendix D - Risk matrix

We have used the following risk matrix:

Consequence Probability	High	Medium	Low
High	Critical	Critical	Medium
Medium	Critical	Medium	Low
Low	Medium	Low	Low

Risk Condition	Consequence	Cat	Risk	Mitigation	Monitoring	Management
Description of the risk	What the direct consequences of the event happening		The combined risk an event puts on the project	How can we avoid or reduce the risk?	What factors can we track that will enable us to determine if the risk is becoming more or less likely?	What contingency plans do we have if the risk becomes a reality
Personnel shortfalls (dropping course/long term sickness etc.)	As the group only consists of three people, a shortage of one would be fatal for the project.		Medium	Monitor and ensure motivation among group members. Personal reasons/sickness can not be accounted for.	Ask group members on a weekly basis what they feel about the subject and project.	Workload have to be distributed on the remaining group members. Customer have to meet for a reorganization of the project. Reduce scope.
Failure to acquire the required information to implement the solution our customer wants.	Immediate: Creating wrong functions. Long term: Unable meet the customers requirements.		Medium	By using an agile development methodology which focuses on transparency and	Getting frequent feedback from the customer; by iterative, incremental deliveries.	Immediate consolidation with the customer, and agree on a changed/reduced scope.

				communication with the customer.		
Unrealistic schedules	Not being able to deliver the product with the constraints set by the customer.		Medium	Plan pessimistically, and make room for Scrum routines during the project. Break tasks into small pieces and estimate each piece.	Emphasize on keeping routines like 'Sprint Planning', 'Daily Scrum' and 'Backlog Grooming' intact.	Draw a line in the sprint backlog of what functionality that must be dropped in order to complete as much as possible of the high-priority goals, and throw the rest back in the backlog at the backlog grooming.
Temporarily sickness: One or more team members are not able to meet or participate for a considerable amount of time (up to 25% or ~7 days of the project span)	Workload on remaining group members will increase.		Critical	Sickness can hardly be avoided on an organizational level. But good health and avoiding overtime work should be enforced within the group.	Track how the other group members are performing and how their work routines are. It is better to leave early than not showing up the next day.	Keep the sick person away from the rest of the group. One can not expect a person recovering from illness to catch up, so work would have to be distributed on the remaining (healthy) group members.
Information redundancy: Failing to agree on what tools we are to use for documentation and project management.	Could result in information redundancy and unnecessary work.		Medium	Agree upon a set of tools early in the project and stick to them. Make sure everyone is familiarized and comfortable	Be aware of each others work routines, and sit together whenever possible. Enforce, talk and repeat about what platform	If a tool/platform is deemed not suitable, reconsider alternatives and agree as a group if a transition to another is worth it.

				with using the tools.	different pieces of information should be stored on.	
Team member not performing due to lack of motivation.	Workload on remaining group members will increase.		Medium	Make sure every group member has a meaningful responsibility, and gets a sense of ownership to the product.	Measure the motivation among group members by asking how and what they feel about their role and the project on a regular basis.	Include all group members on an activity not directly related to the project. Reconsider their respective responsibility and listen to what they think would make them thrive better.
The group is having trouble communicating in a swift and efficient way.	Potential wasted work hours, uncertainty within the group.		Medium	Decide early on a communication tool that is not bound to one device. Have alternative channels to reach each other if one is not responding.	Look at, and think about the overall satisfaction with the tool currently being used. Are people making use of the tool? Are there a lot of confusion?	If a tool/platform is deemed not suitable, reconsider alternatives and agree as a group if a transition to another is worth it.
Work is lost due to hardware faults/accidents.	Project is set back due to lost resources.		Critical	Use tools from companies that have well established routines for backup. Version control. Enforce personal backup as well.	Be aware of each others work routines, and sit together whenever possible.	Get a sense of the scale of work that has been lost. If possible, recreate simplified versions of the work that has been lost. But always inform the customer. Reduce scope if necessary.

Hardware faults. A group member is put out of action because it does not have the tools necessary to work.	Work hours are lost. Work distribution may change.		Medium	Accidents can not be controlled, but make sure no one is too dependent on their 'own' hardware to function.	Use Daily Scrums and share what problems you have encountered.	Use terminals on campus. Redistribute work in order for everyone to function optimally with the tools they currently have access to..